24591: 中序表达式转后序表达式

stack, http://cs101.openjudge.cn/practice/24591/

思路：

用oi表示括号层数，用compare排列优先级别，更优先的更早输出

for \_ in range(int(input())):

stack=[]

output=[]

i=input().strip()

oi=0

ind=0

compare={"+":1,"-":1,"\*":2,"/":2}

while ind < len(i):

r=i[ind]

if r =="(":

oi+=1

elif r.isdigit():

for t in range(ind,len(i)):

if i[t].isdigit() or i[t]==".":

if t==len(i)-1:

t=len(i)

break

continue

break

output.append(i[ind:t])

ind=t

continue

elif r==")":

while stack and oi==stack[-1][-1]:

output.append(stack.pop()[0])

oi-=1

else:

while stack and compare[stack[-1][0]]>=compare[r] and oi == stack[-1][-1]:

output.append(stack.pop()[0])

stack.append((i[ind],oi))

ind+=1

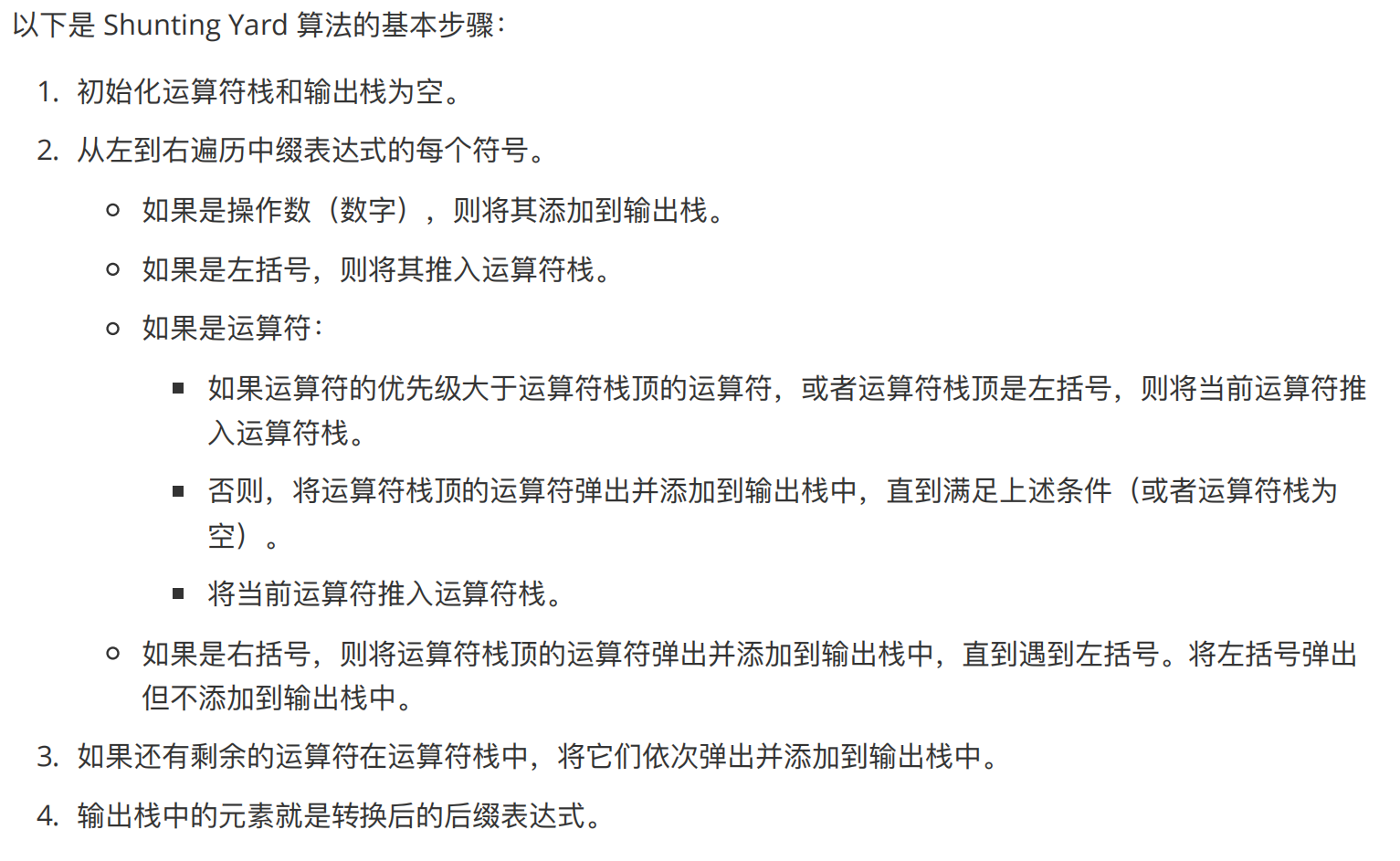
while stack:

sm=stack.pop()

output.append(sm[0])

print(" ".join(map(str,output)))

调度场算法



def inp(s):

#s=input().strip()

import re

s=re.split(r'([\(\)\+\-\\*\/])',s)

s=[item for item in s if item.strip()]

return s

exp = "(3)\*((3+4)\*(2+3.5)/(4+5)) "

print(inp(exp))

类似于栈的**重复使用**表达储存室path:

[131. 分割回文串 - 力扣（LeetCode）](https://leetcode.cn/problems/palindrome-partitioning/solutions/2059414/hui-su-bu-hui-xie-tao-lu-zai-ci-pythonja-fues/)

class Solution:

def partition(self, s: str) -> List[List[str]]:

n = len(s)

ans = []

**path = []**

# 考虑 s[i:] 怎么分割

def dfs(i: int) -> None:

if i == n: # s 分割完毕

ans.append(path.copy()) # 复制 path

return

for j in range(i, n): # 枚举子串的结束位置

t = s[i: j + 1] # 分割出子串 t

if t == t[::-1]: # 判断 t 是不是回文串

path.append(t)

# 考虑剩余的 s[j+1:] 怎么分割

dfs(j + 1)

path.pop() # 恢复现场

dfs(0)

return ans

由于使用有优先级，优先的使用完低优先的使用，使用同时减少了对已经做好的讨论，牛而逼之

[2680. 最大或值 - 力扣（LeetCode）](https://leetcode.cn/problems/maximum-or/solutions/2268795/tan-xin-qian-hou-zhui-fen-jie-pythonjava-wrv1/)

用前缀后缀的形式构造

class Solution:

def maximumOr(self, nums: List[int], k: int) -> int:

n = len(nums)

# suf[i] 表示 nums[i+1:] 的 OR

suf = [0] \* n

for i in range(n - 2, -1, -1):

suf[i] = suf[i + 1] | nums[i + 1]

# pre 表示 nums[:i] 的 OR

ans = pre = 0

for x, suf\_or in zip(nums, suf):

ans = max(ans, pre | (x << k) | suf\_or)

pre |= x

return ans

空间优化

设 nums 所有数的 OR 为 allOr。

能否直接算出从 allOr 中去掉 x=nums[i] 后的结果？把结果与 x << k 计算 OR，就是方法一计算的内容。

可以，做法如下**（NB）**：

1.先通过异或运算，直接去掉 x，即 allOr ^ x。但这样做不一定正确，因为可能其他数在去掉的比特位上是 1。要把这种 1 重新加回来，才是正确结果。

2.如果有多个 nums[i] 在同一个比特位上都是 1，那么去掉 x 后的其余 n−1 个数的 OR，在这个比特位上也是 1。换句话说，无论去掉哪个 x，这些比特位恒为 1。用二进制数 fixed 记录这些恒为 1 的比特位。

3.于是，去掉 x 后的其余 n−1 个数的 OR 等于 (allOr ^ x) | fixed，表示先直接去掉 allOr 中的 x，再通过 fixed 修正。

class Solution:

def maximumOr(self, nums: List[int], k: int) -> int:

all\_or = fixed = 0

for x in nums:

# 如果在计算 all\_or |= x 之前，all\_or 和 x 有公共的 1

# 那就意味着有多个 nums[i] 在这些比特位上都是 1

fixed |= all\_or & x # 把公共的 1 记录到 fixed 中

all\_or |= x # 所有数的 OR

return max((all\_or ^ x) | fixed | (x << k) for x in nums)

练习T02775: 文件结构“图”

http://cs101.openjudge.cn/practice/02775/

这小类，小代码写的真牛逼

# 夏天明，元培学院

from sys import exit

class dir:

def \_\_init\_\_(self, dname):

self.name = dname

self.dirs = []

self.files = []

def getGraph(self):

g = [self.name]

for d in self.dirs:

subg = d.getGraph()

g.extend(["| " + s for s in subg])#完成对下属文件的**多层**“| ”排序输出

for f in sorted(self.files):

g.append(f)

return g

n = 0

while True:

n += 1

**stack = [dir("ROOT")]#这里面全是类的索引，而不是简单的文件夹**

**#这里体现了类在使用时候的优势：可以预设建立多个不同的下属列表，便于增加和改动，用索引随时调用最新版本的类下属列表，同时快速建立不同的地址，增加调用都很便捷**

while (s := input()) != "\*":

if s == "#": exit(0)

if s[0] == 'f':

stack[-1].files.append(s)

elif s[0] == 'd':

stack.append(dir(s))

**stack[-2].dirs.append(stack[-1])#把该文件夹类的索引归到上一个的下属dir里面**

else:

**stack.pop()#它的任务完成了，可以退役了**

print(f"DATA SET {n}:")

print(\*stack[0].getGraph(), sep='\n')#其实只有这一步调用了getGraph函数获得最终输出

print()

## 卡特兰数的应用

通用代码：

n = int(input())

dp = [0] \* (n + 1)

dp[0] = 1

for i in range(1, n + 1):

for j in range(i):

dp[i] += dp[j] \* dp[i - 1 - j]

print(dp[n])

Just like

[OpenJudge - 24755:有多少种二叉树](http://cs101.openjudge.cn/practice/24755/)

[OpenJudge - 27217:有多少种合法的出栈顺序](http://cs101.openjudge.cn/practice/27217/)

以下是卡特兰数的典型问题：

1. 二叉树的形态数目

问题：

n 个节点能构成多少种不同的二叉树？

解释：确定根节点后，左子树有

i 个节点，右子树有n−1−i 个节点，递归求解。

2. 有效的括号组合

问题：

n 对括号能组成多少种合法的嵌套形式？

解释：第一个括号必须为左括号，其内部和右侧的括号组合需满足平衡条件。

3. 凸多边形的三角形划分

问题：将

n+2 条边的凸多边形用不相交的对角线划分为三角形，有多少种方法？

解释：固定一条边作为基边，递归划分左右区域。

4. 栈排序问题

问题：一个栈的进栈序列为

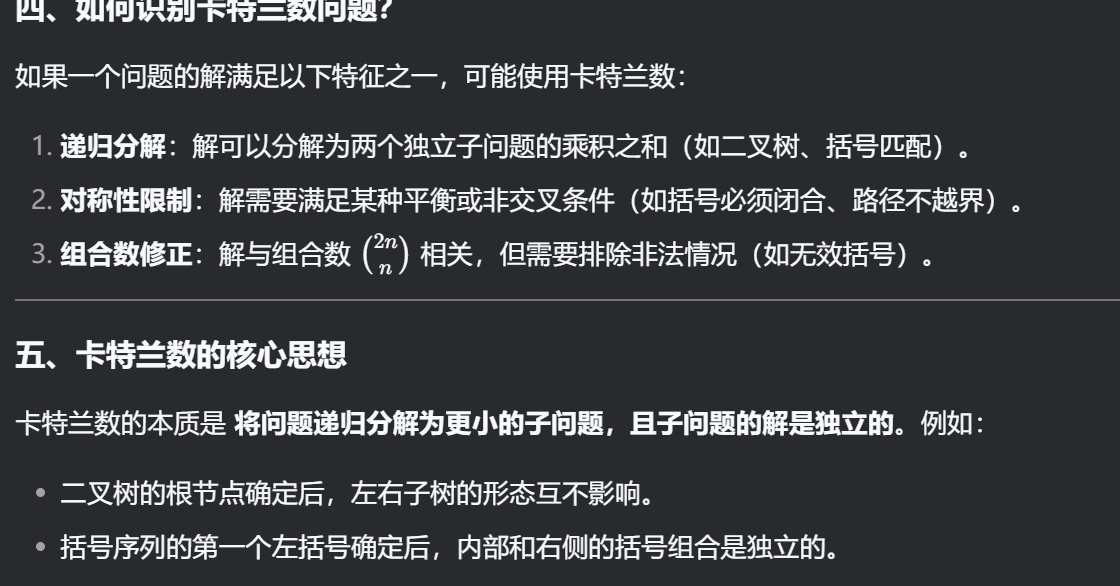
1,2,...,n，有多少种不同的出栈顺序？

解释：每个操作是“进栈”或“出栈”，需保证出栈次数不超过进栈次数。

5. Dyck路径（格点路径）

问题：从

(0,0) 到 (2n,0)，每一步只能向右上或右下走，且不穿过 x 轴的路径数。



def catalan(n):

c = 1

for i in range(n):

c = c \* 2 \* (2 \* i + 1) // (i + 2)

return c

print(catalan(3)) # 输出5（3个节点的二叉树形态数）

Mention：  
class Solution:

def sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:

# 终止条件：空数组

if not nums:

return None

# 计算中间索引

mid = len(nums) // 2

# 创建根节点

root = TreeNode(nums[mid])

# 递归构造左子树和右子树

root.left = **self.**sortedArrayToBST(nums[:mid])

root.right = **self.**sortedArrayToBST(nums[mid+1:])

return root

注意在class中进行调用要用self.--（函数名称），leetcode

路径连通

给你一个整数 n，表示图中的节点数量，这些节点按从 0 到 n - 1 编号。

同时给你一个长度为 n 的整数数组 nums，该数组按 非递减 顺序排序，以及一个整数 maxDiff。

如果满足 |nums[i] - nums[j]| <= maxDiff（即 nums[i] 和 nums[j] 的 绝对差 至多为 maxDiff），则节点 i 和节点 j 之间存在一条 无向边 。

此外，给你一个二维整数数组 queries。对于每个 queries[i] = [ui, vi]，需要判断节点 ui 和 vi 之间是否存在路径。

返回一个布尔数组 answer，其中 answer[i] 等于 true 表示在第 i 个查询中节点 ui 和 vi 之间存在路径，否则为 false。

class Solution:

def pathExistenceQueries(self, n: int, nums: List[int], maxDiff: int, queries: List[List[int]]) -> List[bool]:

**解法一：记录间断点**

idx = [i for i, (x, y) in enumerate(pairwise(nums)) if y - x > maxDiff]

return [bisect\_left(idx, u) == bisect\_left(idx, v) for u, v in queries]

**解法二：记录所属区间**

id = [0] \* n # 每个节点所在连通块的编号

for i in range(1, n):

id[i] = id[i - 1]

if nums[i] - nums[i - 1] > maxDiff:

id[i] += 1 # 找到了一个新的连通块

return [id[u] == id[v] for u, v in queries]

**Dijkstra一**定要用set的visited，不能按照自己的思路在路径stack里面储存！！！

p=int(input())

class node:

def \_\_init\_\_(self,name):

self.name=name

self.go={}

mapi={}

for j in range(p):

s=input().strip()

mapi[s]=node(s)

q=int(input())

for g in range(q):

a,b,d=map(str,input().split())

d=int(d)

mapi[a].go[b]=d

mapi[b].go[a]=d

r=int(input())

import heapq

def pri(a):

s=[]

out=""

for i in range(1,len(a)):

s.append(f"->({mapi[a[i]].go[a[i-1]]})->")

for l in range(len(a)-1):

out+=a[l]+s[l]

out+=a[-1]

return out

for \_\_ in range(r):

stack=[]

havegone=set()

a,b=map(str,input().split())

havegone.add((a,b))

heapq.heappush(stack,(0,a,[a]))

while stack:

step,m,path=heapq.heappop(stack)

if m==b:

print(pri(path))

break

elif m in havegone:

continue

else:

havegone.add(m)注意时间，不能在j里面入havegone，要在弹出来时候havegone确保该时候是最小时间到达点

for j,kl in mapi[m].go.items():

if j not in havegone:

heapq.heappush(stack,(step+kl,j,path+[j]))

[3362. 零数组变换 III - 力扣（LeetCode）](https://leetcode.cn/problems/zero-array-transformation-iii/description/)

太牛逼了，无论是思路上（动态最左的思路）：当nums[i] 不等于0，这时候要想删除掉i，贪心方面考虑就应该优先选择到达最右边的“消除”

还是写法上：heap维持到要用的时候而不是对每个index建立可用的heap（减少了内存使用而且不用标志haveused），用diff列表表示截至位置、使其无法使用的思路，用sum\_d表示i处剩余能用的消除的思路简直无可挑剔。

class Solution:

    def maxRemoval(self, nums: List[int], queries: List[List[int]]) -> int:

        queries.sort(key=lambda q: q[0])  # 按照左端点从小到大排序

        h = []

        diff = [0] \* (len(nums) + 1)

        sum\_d = j = 0

        for i, x in enumerate(nums):

            sum\_d += diff[i]

            # 维护左端点 <= i 的区间

            while j < len(queries) and queries[j][0] <= i:

                heappush(h, -queries[j][1])  # 取相反数表示最大堆

                j += 1

            # 选择右端点最大的区间

            while sum\_d < x and h and -h[0] >= i:

                sum\_d += 1

                diff[-heappop(h) + 1] -= 1

            if sum\_d < x:

                return -1

        return len(h)